SQL: Structured Query Language
Data Manipulation Language (DML) – Part II

CSC343, Introduction to Databases
Nosayba El-Sayed (based on slides from Diane Horton)
Fall 2015
SQL Recap

SELECT sID, AVG(grade) as stdavg
FROM Took
WHERE sID > 22222
GROUP BY sID
HAVING AVG(grade) > 70
ORDER BY AVG(grade) DESC

If you’re not using **DISTINCT** or **GROUPING**, you can ORDER BY UN-SELECTED attributes.

Order of execution:
1. FROM
2. ON
3. JOIN
4. OUTER
5. WHERE
6. GROUP BY
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
SQL Recap

**Order of execution:**
1. **FROM**
2. **ON**
3. **JOIN**
4. **OUTER**
5. **WHERE**
6. **GROUP BY**
7. **HAVING**
8. **SELECT**
9. **DISTINCT**
10. **ORDER BY**

**SELECT** sID
**FROM**  Student
**WHERE** sID > 22222
**ORDER BY**  cgpa  DESC

If you’re not using **DISTINCT** or **GROUPING**, you can **ORDER BY** **UN-SELECTED** attributes.
SQL Joins
The joins you know from RA

<table>
<thead>
<tr>
<th>Expression within SQL statements</th>
<th>Meaning (RA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R, S )</td>
<td>R ( \times ) S</td>
</tr>
<tr>
<td>( R ) cross join ( S )</td>
<td>R ( \bowtie ) S</td>
</tr>
<tr>
<td>( R ) natural join ( S )</td>
<td>R ( \bowtie ) condition S</td>
</tr>
<tr>
<td>( R ) join ( S ) on &lt;Condition&gt;</td>
<td>R ( \bowtie ) condition S</td>
</tr>
<tr>
<td>( R ) natural left [outer] join ( S )</td>
<td>R ( \bowtie ) S</td>
</tr>
<tr>
<td>( R ) natural right [outer] join ( S )</td>
<td>R ( \bowtie ) S</td>
</tr>
<tr>
<td>( R ) natural full [outer] join ( S )</td>
<td>R ( \bowtie ) S</td>
</tr>
</tbody>
</table>
The joins you know from RA (Cont.)

<table>
<thead>
<tr>
<th>Expression within SQL statements</th>
<th>Meaning (RA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R <strong>left</strong> [outer] join S on &lt;cond&gt;</td>
<td>R ( \nabla )_{\text{condition}} S</td>
</tr>
<tr>
<td>R <strong>right</strong> [outer] join S on &lt;cond&gt;</td>
<td>R ( \nabla )_{\text{condition}} S</td>
</tr>
<tr>
<td>R <strong>full</strong> [outer] join S on &lt;cond&gt;</td>
<td>R ( \nabla )_{\text{condition}} S</td>
</tr>
</tbody>
</table>
Dangling tuples

• With joins that require some attributes to match, tuples lacking a match are left out of the results.
• We say that they are “dangling”.
• An outer join preserves dangling tuples by padding them with NULL in the other relation.
• A join that doesn’t pad with NULL is called an inner join.
Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

R NATURAL LEFT JOIN S

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

R NATURAL RIGHT JOIN S

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>NULL</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Example

R NATURAL FULL JOIN S

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
In practice natural join is dangerous

• Attributes with matching names don’t necessarily mean matching meanings!
• Having implicit comparisons impairs readability.
• Also: if the schema changed, a query that looks fine may actually be broken, without being able to tell.
• Best practise: Don’t use natural join.
Summary of join expressions

- **Cartesian product**
  \[ A \text{ cross join } B \text{ same as } A, B \]

- **Theta-join**
  \[ A \text{ join } B \text{ on } C \text{ no padding of tuples} \]
  \[ A \text{ left|right|full join } B \text{ on } C \text{ padding} \]

- **Natural join**
  \[ A \text{ natural join } B \text{ no padding of tuples} \]
  \[ A \text{ natural left|right|full join } B \text{ padding} \]
Keywords INNER and OUTER

• There are keywords INNER and OUTER, but you never need to use them.

• Your intentions are clear anyway:
  • You get an OUTER join iff you use the keywords LEFT, RIGHT, or FULL.
  • If you don’t use the keywords LEFT, RIGHT, or FULL you get an INNER join.
Impact of having null values

So, what's a NULL value??
Missing Information

• Two common scenarios:
  
  • **Missing value.**
    
    E.g., we know a student has some *email* address, but we don’t know what it is.
  
  • **Inapplicable attribute.**
    
    E.g., the value of attribute *spouse* for an unmarried person.
Representing missing information

- One possibility: use a special value as a placeholder. E.g.,
  - If age unknown, use -1.
  - If StNum unknown, use 999999999.

- Pros and cons?

- Better solution: use a value not in any domain. We call this a **null value**.

- Tuples in SQL relations can have **NULL** as a value for one or more components.
Checking for null values

• You can compare an attribute value to NULL with
  • IS NULL
  • IS NOT NULL

• Example:
  ```sql
  SELECT * 
  FROM Course 
  WHERE breadth IS NULL;
  ```

• Note: do not use WHERE breadth = NULL;
Impact of null values on SQL expressions?

• Arithmetic expressions?
  - Result is always NULL
  - Example: \((x + \text{grade}) \rightarrow \text{NULL}\)
  - Even if ‘grade’ is 0!
    - i.e. \((x \times 0) \rightarrow \text{NULL}\)
  - Also: \((x - x) \rightarrow \text{NULL}\)

• Comparison operators? (\(>\), \(<\), \(=\), ...)?
  - \((x < 32) \rightarrow \text{UNKNOWN}\)
  - Result is UNKNOWN
  - This UNKNOWN is a truth-value!
  - Truth-values in SQL are: (TRUE, FALSE, UNKNOWN)

• What if we have: WHERE \((x<32 \text{ OR name}='\text{Alex}')\)?
Evaluating Logic Expressions with **UNKNOWN**

- **Logic**: TRUE, FALSE, UNKNOWN
  - UNKNOWN OR FALSE → UNKNOWN
  - UNKNOWN OR TRUE → TRUE
  - UNKNOWN AND TRUE → UNKNOWN
  - UNKNOWN AND FALSE → FALSE
  - NOT UNKNOWN → UNKNOWN

- **In SQL**
  - A tuple is in a query result **iff** the result of the WHERE clause is **TRUE**
  - **Demo**: *where-null*

**Ternary logic tricks:**

- TRUE = 1
- FALSE = 0
- UNKNOWN = ½
- AND = min(…)
- OR = max(…)
- NOT = 1-x
### Thinking of the truth-values as numbers

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>as nums</th>
<th>A and B</th>
<th>min</th>
<th>A or B</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>1, 1</td>
<td>T</td>
<td>1</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>TF or FT</td>
<td>1, 0</td>
<td>F</td>
<td>0</td>
<td>T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>0, 0</td>
<td>F</td>
<td>0</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>TU or UT</td>
<td>1, 0.5</td>
<td>U</td>
<td>0.5</td>
<td>T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>FU or UF</td>
<td>0, 0.5</td>
<td>F</td>
<td>0</td>
<td>U</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>0.5, 0.5</td>
<td>U</td>
<td>0.5</td>
<td>U</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Impact of null values on aggregation

• “Aggregation ignores $\text{NULL}$.”

• $\text{NULL}$ never contributes to a sum, average, or count, and can never be the minimum or maximum of a column (unless every value is $\text{NULL}$).

• If ALL values are $\text{NULL}$ in a column, then the result of the aggregation is $\text{NULL}$.
  • Exception: $\text{COUNT}$ of an empty set is 0.
Impact of null values on aggregation

• **COUNT()**
  - COUNT(R.*) = 2  \[\text{COUNT}(R.x) = 1\]
  - COUNT(S.*) = 1  \[\text{COUNT}(S.x) = 0\]
  - COUNT(T.*) = 0  \[\text{COUNT}(T.x) = 0\]

• **Other aggregations** (e.g. **MIN/MAX**)
  - MIN(R.x) = 1  \[\text{MAX}(R.x) = 1\]
  - MIN(S.x) = NULL  \[\text{MAX}(S.x) = NULL\]
  - MIN(T.x) = NULL  \[\text{MAX}(T.x) = NULL\]
## Summary

<table>
<thead>
<tr>
<th></th>
<th>Some nulls in A</th>
<th>All nulls in A</th>
</tr>
</thead>
<tbody>
<tr>
<td>min(A)</td>
<td>ignore the nulls</td>
<td>NULL</td>
</tr>
<tr>
<td>max(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum(A)</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>avg(A)</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>count(A)</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>count(*)</td>
<td>all tuples count</td>
<td></td>
</tr>
</tbody>
</table>
Subqueries

BRACE YOURSELF

SHIT JUST GOT REAL
Subqueries in a **FROM** clause

- Instead of a *relation* name in the **FROM** clause, we can use a *subquery*.
- The subquery must be parenthesized.
- **Must name the result**, so you can refer to it in the outer query.
Example

• What does this do?
  
  ```sql
  SELECT sid, dept||cnum as course, grade
  FROM Took,
  (SELECT *
   FROM Offering
   WHERE instructor='Horton') Hofferings
  WHERE Took.oid = Hofferings.oid;
  ```

• This FROM is analogous to:
  ```sql
  Took × ρHofferings («subquery»)
  ```

• Can you suggest another version?
Subquery as a value in a **WHERE**

- If a subquery is guaranteed to produce exactly one tuple, then the subquery can be used as a **value**.
- Simplest situation: that one tuple has only one attribute.
Example

• Find all students with a cgpa greater than that of student 99999.

SELECT sid, surname
FROM Student
WHERE cgpa >
(SELECT cgpa
  FROM Student
  WHERE sid = 99999);
Special cases

• What if the subquery returns **NULL**?
  • Evaluates to UNKNOWN, tuple not returned

• What if the subquery could return more than one value?

• When a subquery can return multiple values, we can make comparisons using a quantifier:
  • cgpa > at least one of them (ANY)
  • cgpa > all of them (ALL)
The Operator ANY

• Syntax:
  \[ x \ «comparison» \text{ANY} («subquery») \]
  or equivalently
  \[ x \ «comparison» \text{SOME} («subquery») \]

• Semantics:
  Its value is true iff the comparison holds for at least one tuple in the subquery result, i.e.,
  \[ \exists y \in «subquery\ results» \mid x \ «comparison» y \]

• \( x \) can be a list of attributes, but this feature is not supported by psql.
The Operator ALL

- **Syntax:**
  \[ x \ «comparison» \text{ ALL («subquery»)} \]

- **Semantics:**
  Its value is true iff the comparison holds for every tuple in the subquery result, i.e.,
  \[ \forall y \in \text{ «subquery results»} | x \ «comparison» y \]

- \(x\) can be a list of attributes, but this feature is not supported by psql.

**Demo:** any and all
The Operator IN

• Syntax:
  \[ x \text{ IN} \ (\text{«subquery»}) \]

• Semantics:
  Its value is true iff \( x \) equals at least one of the tuples in the subquery result.

• \( x \) can be a list of attributes, and psql does support this feature.
Example – Q2 in Class Exercises

What does this do?

```
SELECT sid, dept||cnum AS course, grade
FROM Took NATURAL JOIN Offering
WHERE
  grade >= 80 AND
  (cnum, dept) IN (SELECT cnum, dept
                      FROM Took NATURAL JOIN Offering
                               NATURAL JOIN Student
                      WHERE surname = 'Lakemeyer');
```
Q3 in Class Exercises

Suppose we have tables R(a, b) and S(b, c).

1. What does this query do?

   ```sql
   SELECT a
   FROM R
   WHERE b IN (SELECT b FROM S);
   ```

2. Can we express this query without using IN?
The Operator EXISTS

• Syntax:
  EXISTS «subquery»

• Semantics:
  Its value is true iff the subquery has at least one tuple.
Example (Q4) : NOT EXISTS

What does this do?

SELECT instructor
FROM Offering Off1
WHERE NOT EXISTS (
  SELECT *
  FROM Offering
  WHERE
  WHERE
  oid <> Off1.oid AND
  instructor = Off1.instructor
);
Scope

- Queries are evaluated from the inside out.
- If a name might refer to more than one thing, use the most closely nested one.
- If a subquery refers only to names defined inside it, it can be evaluated once and used repeatedly in the outer query.
- If it refers to any name defined outside of itself, it must be evaluated once for each tuple in the outer query.

These are called correlated subqueries.
Renaming can make scope explicit

```
SELECT instructor
FROM Offering Off1
WHERE NOT EXISTS ( 
    SELECT *
    FROM Offering Off2
    WHERE
        Off2.oid <> Off1.oid AND
        Off2.instructor = Off1.instructor
);```
Summary: where subqueries can go

• As a relation in a FROM clause.
• As a value in a WHERE clause.
• With ANY, ALL, IN or EXISTS in a WHERE clause.
• As operands to UNION, INTERSECT or EXCEPT.
• Reference: textbook, section 6.3.
Q5

- For each course find the instructor who has taught the most offerings of it. If there are ties, include them all. Report the course (e.g., csc343), instructor, and number of offerings of the course by that instructor. Use one or more views to hold intermediate steps.

ROS1 Schema

Students(sID, surName, campus)
Courses(dept, cNum, name, breadth)
Offerings(oID, dept, cNum, term, instructor)
Took(sID, oID, grade)